Pneumonia detection

Introduction

Import necessary libraries

```
%%time

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import cv2, os, random
import plotly
import plotly.graph_objs as go
import plotly.express as px
from plotly.offline import init_notebook_mode, plot, iplot

#____________________________________________________________

import glob
import tensorflow
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from mlxtend.plotting import plot_confusion_matrix
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.applications.vgg16 import VGG16
from sklearn.model_selection import train_test_split

#____________________________________________________________

from tqdm.notebook import tqdm
from termcolor import colored
import albumentations as A

#____________________________________________________________

from warnings import filterwarnings
filterwarnings("ignore")

from sklearn import set_config
set_config(print_changed_only = False)
directory = "chest-xray-pneumonia/chest_xray/"

#
```

"

Load the datasets

```
%%time

train_df = glob.glob("chest-xray-pneumonia/chest_xray/train/**/*.jpeg")
test_df = glob.glob(".chest-xray-pneumonia/chest_xray/test/**/*.jpeg")
validation_df = glob.glob("/chest-xray-pneumonia/chest_xray/val/**/*.jpeg")

print(colored("The datasets were succesfully loaded...", color = "green", attrs = ["bold", "dark"]))
```

Look at train and test sets

```
train_df[:5], test_df[:5]
```

```
([' ../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person1180_virus_2010.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person1230_virus_2081.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person1513_virus_2632.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person124_virus_238.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person746_virus_1369.jpeg'],
 ['../input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA/person1676_virus_2892.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA/person1650_virus_2852.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA/person22_virus_55.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA/person122_bacteria_582.jpeg',
  '../input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA/person85_bacteria_417.jpeg'])
```

How many images are in each dataset?

```
print("There is {} images in the training dataset".format(len(train_df)))
print("There is {} images in the test dataset".format(len(test_df)))
print("There is {} images in the validation dataset".format(len(validation_df)))
```

```
There is 5216 images in the training dataset
There is 624 images in the test dataset
There is 16 images in the validation dataset
```

How many of the pictures are of pneumonic lungs and how many are of normal lungs

```
datasets, pneumonia_lung, normal_lung = ["train", "test", "val"], [], []

for i in datasets:
    path = os.path.join(directory, i)
    normal = glob.glob(os.path.join(path, "NORMAL/*.jpeg"))
    pneumonia = glob.glob(os.path.join(path, "PNEUMONIA/*.jpeg"))
```

```
    normal_lung.extend(normal), pneumonia_lung.extend(pneumonia)

print("The number of pneumonia images is {}".format(len(pneumonia_lung)))
print("The number of non-pneumonia images is {}".format(len(normal_lung)))
```

```
The number of pneumonia images is 4273
The number of non-pneumonia images is 1583
```

Shuffle the images

```
random.shuffle(normal_lung)
random.shuffle(pneumonia_lung)
images = normal_lung[:50] + pneumonia_lung[:50]
images[:10]
```

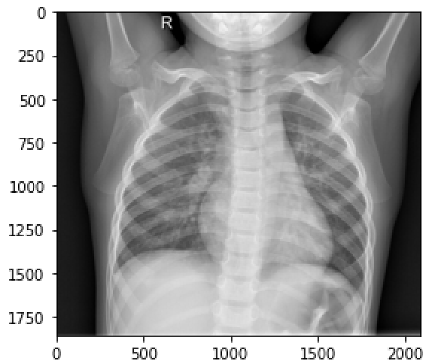
```
['../input/chest-xray-pneumonia/chest_xray/test/NORMAL/IM-0079-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0379-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0673-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/NORMAL2-IM-0839-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/NORMAL2-IM-1162-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/NORMAL2-IM-0843-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0242-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0355-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/NORMAL2-IM-1285-0001.jpeg',
 '../input/chest-xray-pneumonia/chest_xray/train/NORMAL/NORMAL2-IM-1385-0001.jpeg']
```

View the images in X-ray format

X-ray imaging creates pictures of the inside of a body. The images show the parts of a body in different shades of black and white. This is because different tissues absorb different amounts of radiation. Calcium in bones absorbs x-rays the most, so bones look white

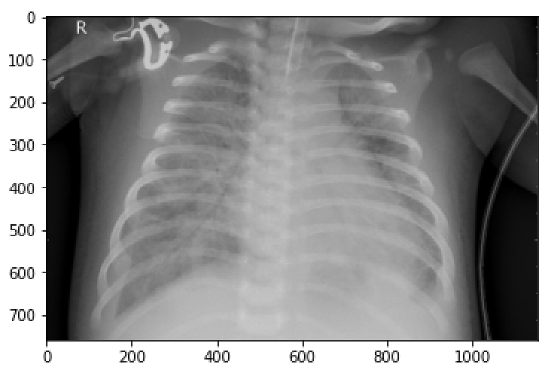
```
normal_lung_image = load_img("/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0115-0001.jpeg")
print("NORMAL")
plt.imshow(normal_lung_image)
plt.show()
```

```
NORMAL
```



```
normal_lung_image = load_img("chest-xray-pneumonia/chest_xray/train/PNEUMONIA/person1000_bacteria_2931.jpeg")
print("PNEUMONIA")
plt.imshow(normal_lung_image)
plt.show()
```

```
PNEUMONIA
```



```
fig = plt.figure(figsize = (20, 15))
columns, rows = 3, 3
for i in range(1, 10):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
```

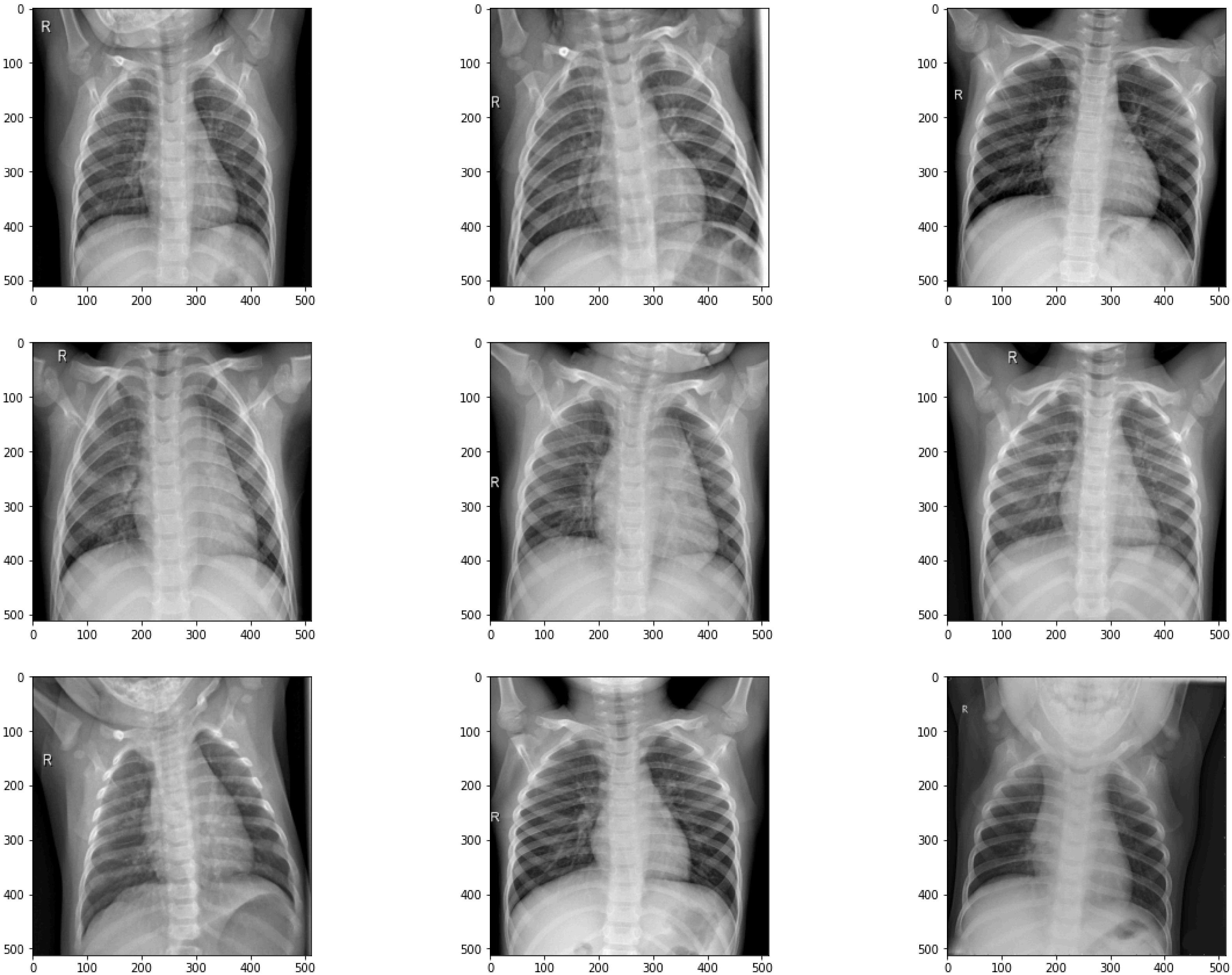


Image erosion

```
fig = plt.figure(figsize = (20, 15))
columns, rows = 3, 3
```

```
for i in range(1, 10):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kernel = np.ones((5, 5), np.uint8)
    image_erosion = cv2.erode(img, kernel, iterations=3)
    fig.add_subplot(rows, columns, i)
    plt.imshow(image_erosion)
```

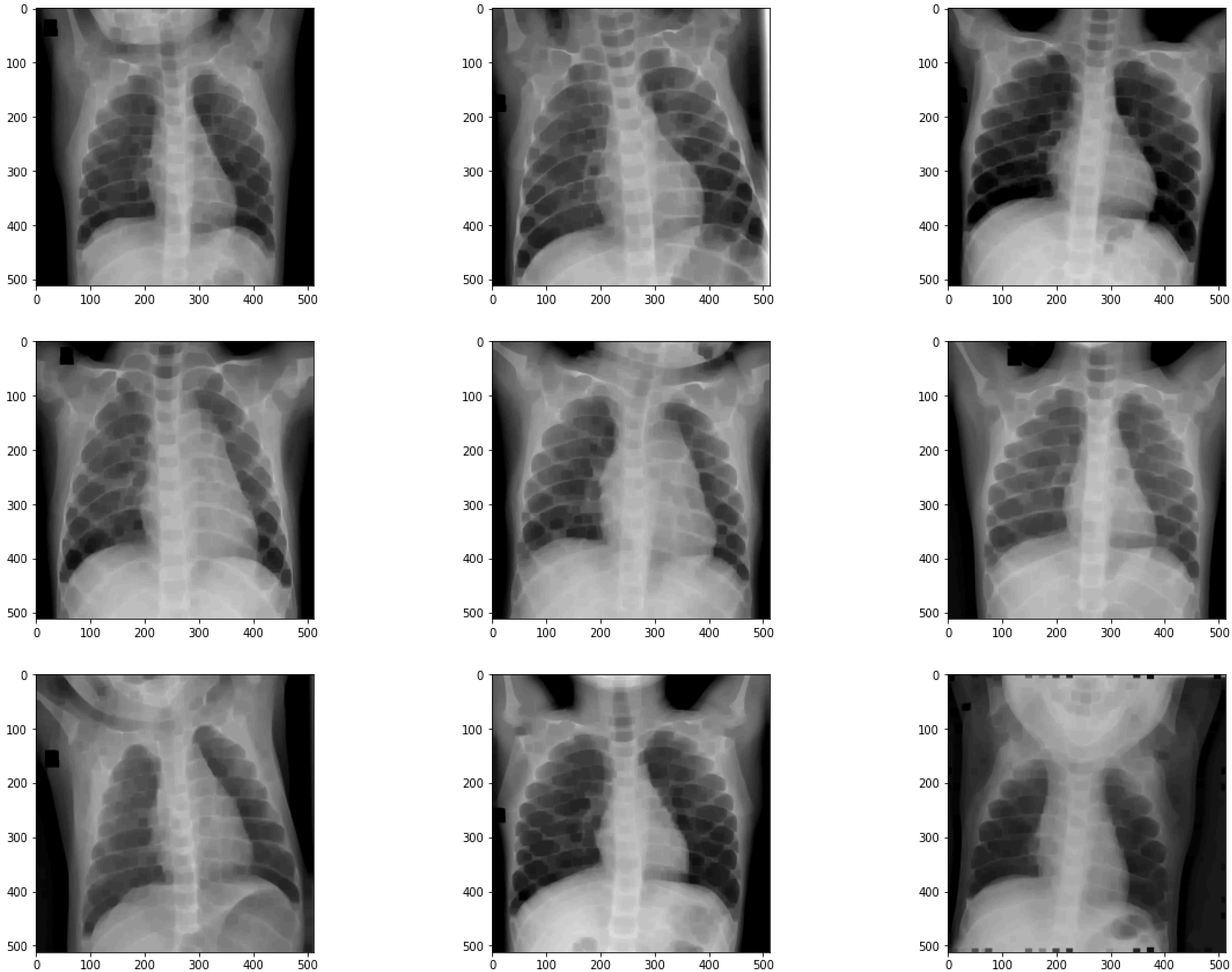
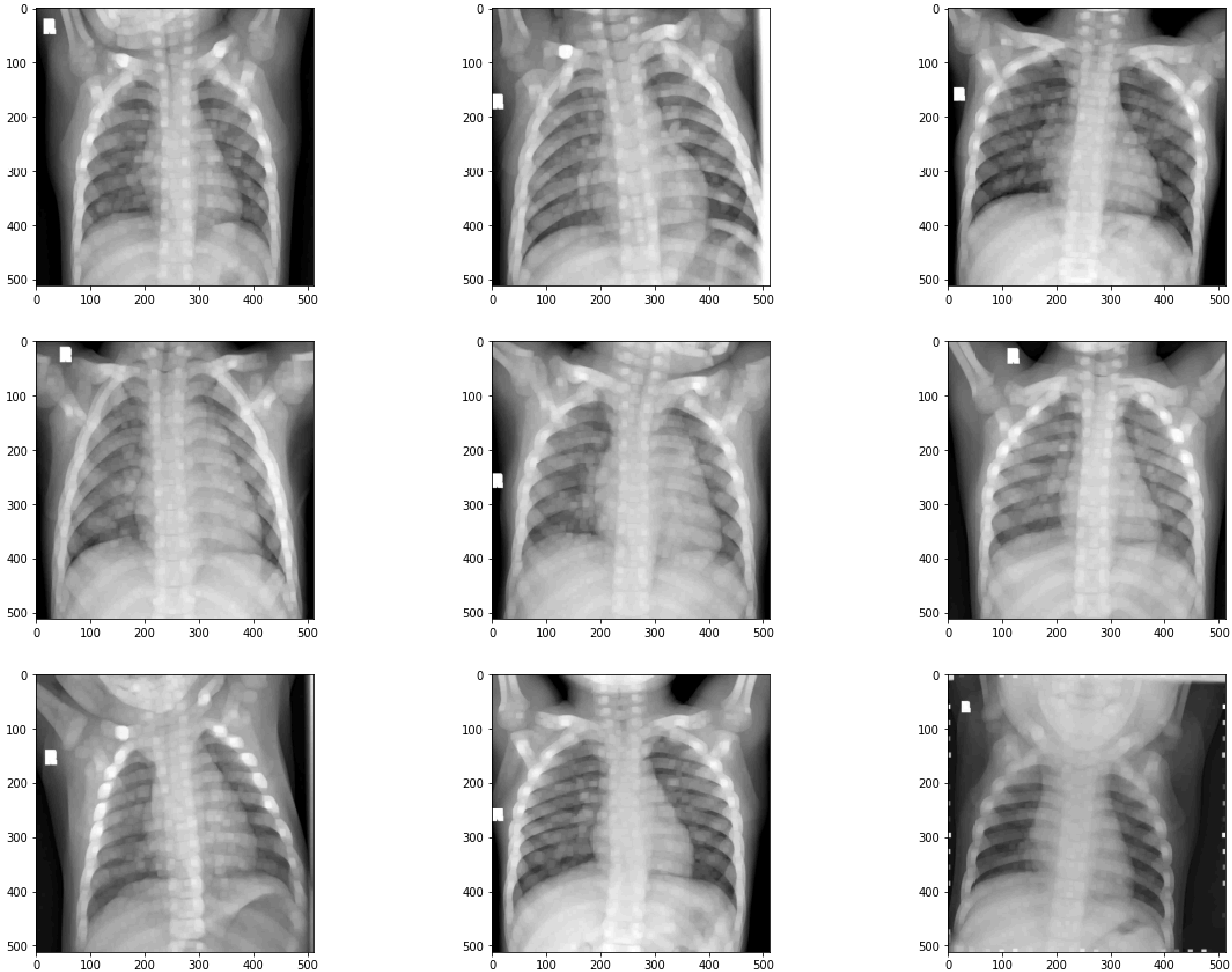


Image dilation

```
fig = plt.figure(figsize = (20, 15))
columns, rows = 3, 3
```

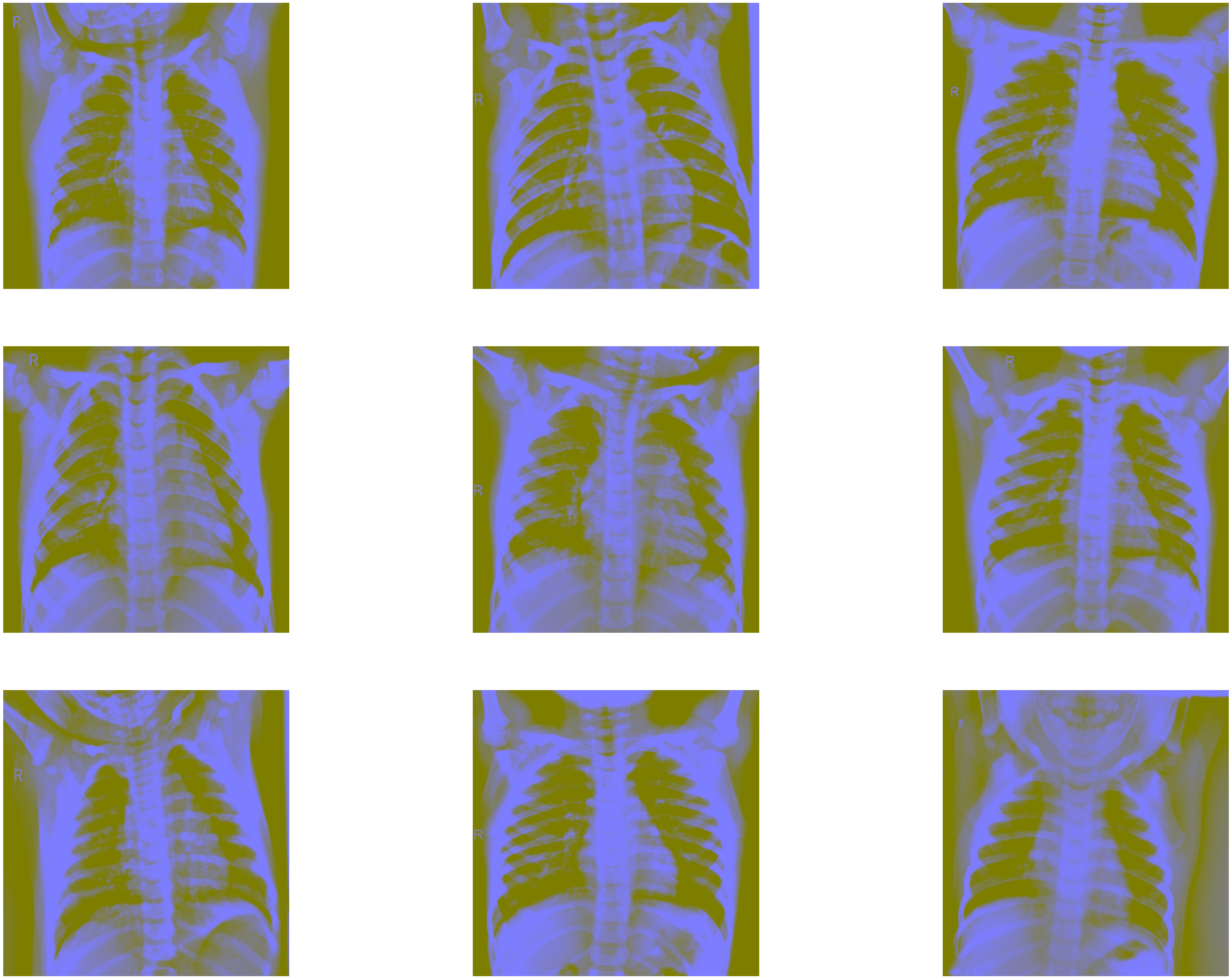
```
for i in range(1, 10):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kernel = np.ones((5, 5), np.uint8)
    image_dilation = cv2.dilate(img, kernel, iterations = 2)
    fig.add_subplot(rows, columns, i)
    plt.imshow(image_dilation)
```



Convert the images to greyscale and then apply Gaussian blur to them

```
fig = plt.figure(figsize = (20, 15))
columns, rows = 3, 3
```

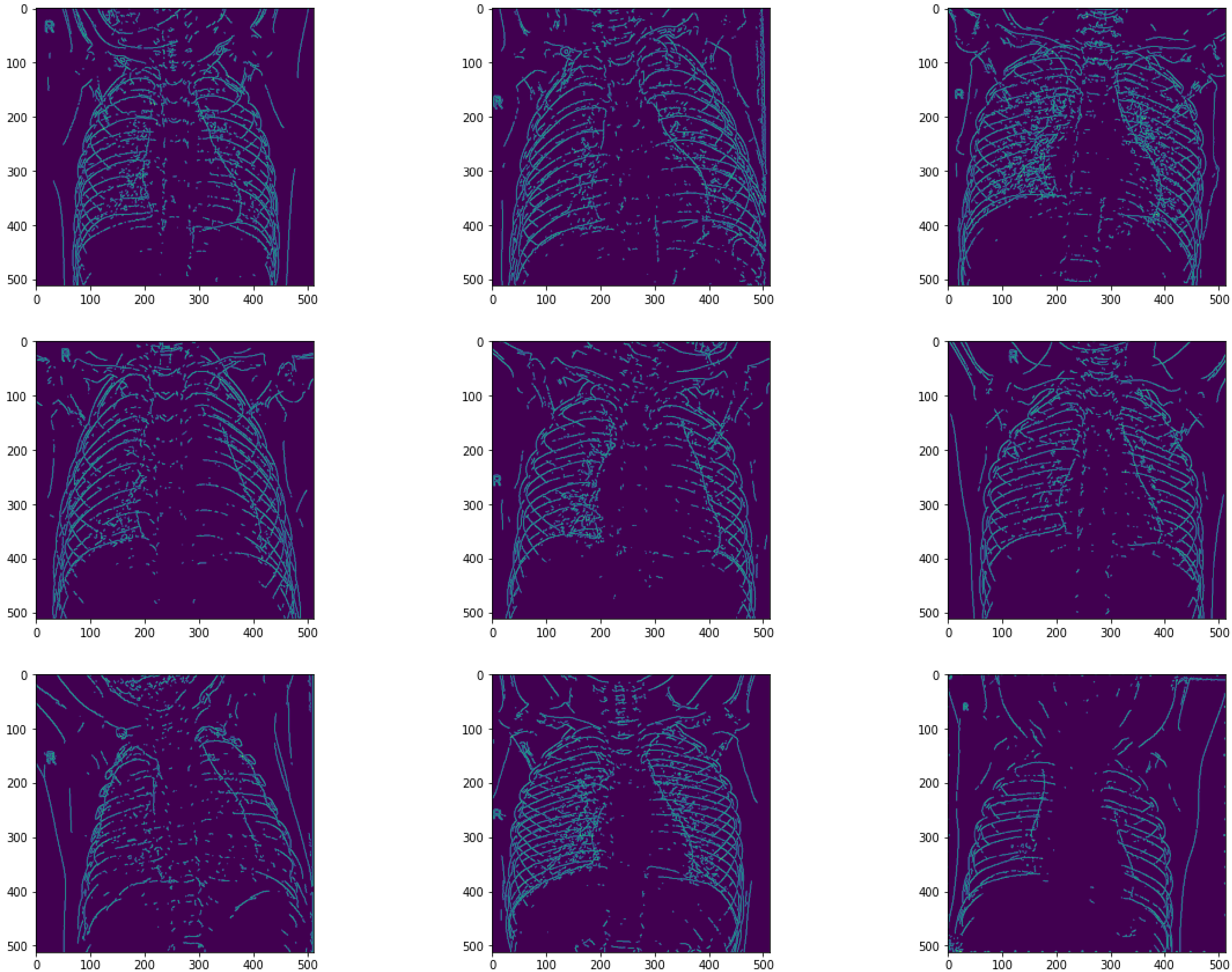
```
for i in range(1, 10):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    img = cv2.addWeighted (img, 4, cv2.GaussianBlur(img, (0, 0), 512/10), -4, 128)
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    plt.axis(False)
```



Canny edge detection:

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems

```
fig = plt.figure(figsize = (20, 15))
columns, rows = 3, 3

for i in range(1, 10):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    detected_edges = cv2.Canny(img, 80, 100)
    fig.add_subplot(rows, columns, i)
    plt.imshow(detected_edges)
```



Build deep learning models

```
train_dir = "chest-xray-pneumonia/chest_xray/train"
test_dir = "chest-xray-pneumonia/chest_xray/test"
```

```
validation_dir = "chest-xray-pneumonia/chest_xray/val"
```

```
%%time

train_datagen = ImageDataGenerator(
            rescale = 1/255.,
            horizontal_flip = True,
            vertical_flip = True,
            rotation_range = 0.3,
            zca_whitening = True,
            width_shift_range = 0.25,
            height_shift_range = 0.25,
            channel_shift_range = 0.35,
            shear_range = 0.2,
            zoom_range = 0.4)

val_test_datagen = ImageDataGenerator(rescale = 1./255)

train_set = train_datagen.flow_from_directory(train_dir, class_mode = "binary", batch_size = 16, target_size = (224, 224))
validation_set = val_test_datagen.flow_from_directory(validation_dir, class_mode = "binary", batch_size = 16, target_size = (224, 224))
test_set = val_test_datagen.flow_from_directory(test_dir, class_mode = "binary", batch_size = 16, target_size = (224, 224))
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
CPU times: user 193 ms, sys: 114 ms, total: 307 ms
Wall time: 2.38 s
```

Cache and prefetch data

If we use flow_from_directory along with ImageDataGenerator() to set up the dataset, it will not be compatible with tensorflow.data.AUTOTUNE. Use tensorflow.keras.preprocessing.image_dataset_from_directory instead to load the dataset.

```
'''
AUTOTUNE = tensorflow.data.experimental.AUTOTUNE

train_set = train_set.cache().prefetch(buffer_size = AUTOTUNE)
test_set = test_set.cache().prefetch(buffer_size = AUTOTUNE)
validation_set = validation_set.cache().prefetch(buffer_size = AUTOTUNE)
'''
pass
```

TRANSFER LEARNING

VGG16 model

```
base_model1 = VGG16(include_top = False, weights = "imagenet", input_shape = (224, 224, 3), pooling = "max",
                    classes = 2)

base_model1.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_max_pooling2d (Global (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

```
model2 = Sequential()
model2.add(base_model1)
model2.add(Flatten())

model2.add(Dense(128, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(32, activation = "relu"))
model2.add(Dense(1, activation = "sigmoid"))

# freeze the layers
for layer in base_model1.layers:
    layer.trainable = False

model2.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])
```

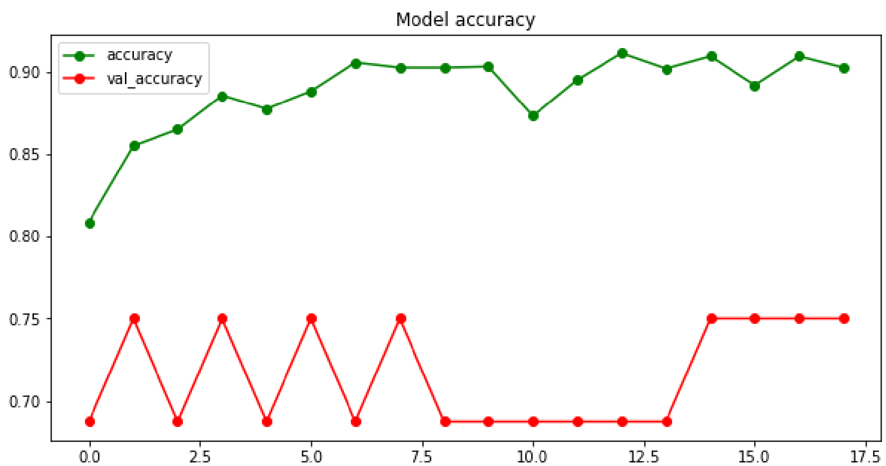
```
%%time

history = model2.fit_generator(train_set, epochs = 20, validation_data = validation_set, steps_per_epoch = 100,
                               callbacks = [early_stopping_callbacks])
```

```
Epoch 1/20
100/100 [==============================] - 38s 358ms/step - loss: 0.3986 - accuracy: 0.8087 - val_loss: 0.6726 - val_accuracy: 0.6875
Epoch 2/20
100/100 [==============================] - 35s 349ms/step - loss: 0.3136 - accuracy: 0.8550 - val_loss: 0.5572 - val_accuracy: 0.7500
Epoch 3/20
100/100 [==============================] - 35s 349ms/step - loss: 0.2985 - accuracy: 0.8650 - val_loss: 0.4124 - val_accuracy: 0.6875
Epoch 4/20
100/100 [==============================] - 35s 346ms/step - loss: 0.2642 - accuracy: 0.8856 - val_loss: 0.6417 - val_accuracy: 0.7500
Epoch 5/20
100/100 [==============================] - 35s 352ms/step - loss: 0.2808 - accuracy: 0.8775 - val_loss: 0.9485 - val_accuracy: 0.6875
Epoch 6/20
100/100 [==============================] - 36s 354ms/step - loss: 0.2444 - accuracy: 0.8881 - val_loss: 0.5148 - val_accuracy: 0.7500
Epoch 7/20
100/100 [==============================] - 35s 348ms/step - loss: 0.2381 - accuracy: 0.9056 - val_loss: 0.9191 - val_accuracy: 0.6875
Epoch 8/20
100/100 [==============================] - 35s 351ms/step - loss: 0.2339 - accuracy: 0.9025 - val_loss: 0.4247 - val_accuracy: 0.7500
Epoch 9/20
100/100 [==============================] - 35s 351ms/step - loss: 0.2245 - accuracy: 0.9025 - val_loss: 0.5990 - val_accuracy: 0.6875
Epoch 10/20
100/100 [==============================] - 35s 347ms/step - loss: 0.2276 - accuracy: 0.9031 - val_loss: 0.5640 - val_accuracy: 0.6875
Epoch 11/20
100/100 [==============================] - 35s 350ms/step - loss: 0.2701 - accuracy: 0.8731 - val_loss: 1.0726 - val_accuracy: 0.6875
Epoch 12/20
100/100 [==============================] - 35s 348ms/step - loss: 0.2345 - accuracy: 0.8950 - val_loss: 0.6455 - val_accuracy: 0.6875
Epoch 13/20
100/100 [==============================] - 35s 347ms/step - loss: 0.2053 - accuracy: 0.9112 - val_loss: 0.6228 - val_accuracy: 0.6875
Epoch 14/20
100/100 [==============================] - 35s 350ms/step - loss: 0.2453 - accuracy: 0.9019 - val_loss: 0.7269 - val_accuracy: 0.6875
Epoch 15/20
100/100 [==============================] - 35s 348ms/step - loss: 0.2235 - accuracy: 0.9094 - val_loss: 1.0353 - val_accuracy: 0.7500
Epoch 16/20
100/100 [==============================] - 35s 350ms/step - loss: 0.2423 - accuracy: 0.8919 - val_loss: 1.0124 - val_accuracy: 0.7500
Epoch 17/20
```

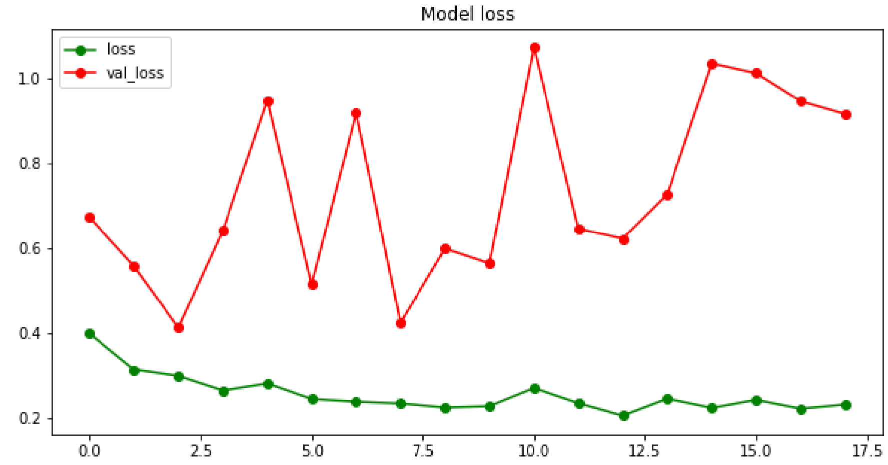
```
100/100 [==============================] - 35s 349ms/step - loss: 0.2217 - accuracy: 0.9094 - val_loss: 0.9464 - val_accuracy: 0.7500
Epoch 18/20
100/100 [==============================] - 35s 351ms/step - loss: 0.2314 - accuracy: 0.9025 - val_loss: 0.9162 - val_accuracy: 0.7500
Restoring model weights from the end of the best epoch.
Epoch 00018: early stopping
CPU times: user 10min 37s, sys: 6.6 s, total: 10min 44s
Wall time: 11min 31s
```

Visualize the performance of model2

```
plt.figure(figsize = (10, 5))
plt.title("Model accuracy")
plt.plot(history.history["accuracy"], "go-")
plt.plot(history.history["val_accuracy"], "ro-")
plt.legend(["accuracy", "val_accuracy"])
plt.show()
```




```
plt.figure(figsize = (10, 5))
plt.title("Model loss")
plt.plot(history.history["loss"], "go-")
plt.plot(history.history["val_loss"], "ro-")
plt.legend(["loss", "val_loss"])
plt.show()
```



Evaluate model2 on the test set

```
#scores = model1.evaluate_generator(test_set)
#print("\n%s: %.3f%%" % (model1.metrics_names[0], scores[0]*100))
#print("\n%s: %.3f%%" % (model1.metrics_names[1], scores[1]*100))

test_loss, test_accuracy = model2.evaluate(test_set, steps = 50)
print("The testing accuracy is: ", test_accuracy * 100, "%")
print("The testing loss is: ", test_loss * 100, "%")
```

```
50/50 [==============================] - 6s 116ms/step - loss: 0.4875 - accuracy: 0.7484
The testing accuracy is:  74.83974099159241 %
The testing loss is:  48.75228703022003 %
```

ResNet50V2 model

```
base_model2 = tensorflow.keras.applications.ResNet50V2(weights = "imagenet",
                                                       input_shape = (224, 224, 3),
                                                       pooling = "max", include_top = False,
                                                       classes = 2)

for layer in base_model2.layers:
    layer.trainable = False

#base_model2.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94674944/94668760 [==============================] - 1s 0us/step
94683136/94668760 [==============================] - 1s 0us/step
```

```
model3 = Sequential()
model3.add(base_model2)
model3.add(Flatten())

model3.add(Dense(128, activation = "relu"))
model3.add(Dense(64, activation = "relu"))
model3.add(Dense(32, activation = "relu"))
model3.add(Dense(1, activation = "sigmoid"))

# freeze the layers
for layer in base_model2.layers:
    layer.trainable = False

model3.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])
```

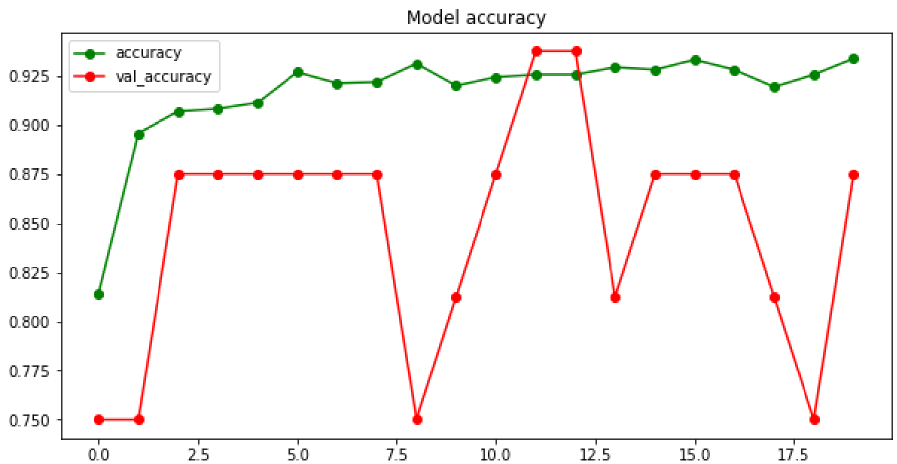
```
%%time

history = model3.fit_generator(train_set, epochs = 20, validation_data = validation_set, steps_per_epoch = 100,
                              callbacks = [early_stopping_callbacks])
```

```
Epoch 1/20
100/100 [==============================] - 39s 358ms/step - loss: 0.4666 - accuracy: 0.8138 - val_loss: 0.5077 - val_accuracy: 0.7500
Epoch 2/20
100/100 [==============================] - 35s 351ms/step - loss: 0.2695 - accuracy: 0.8956 - val_loss: 0.4809 - val_accuracy: 0.7500
Epoch 3/20
100/100 [==============================] - 36s 356ms/step - loss: 0.2172 - accuracy: 0.9069 - val_loss: 0.3471 - val_accuracy: 0.8750
Epoch 4/20
100/100 [==============================] - 35s 346ms/step - loss: 0.2233 - accuracy: 0.9081 - val_loss: 0.2734 - val_accuracy: 0.8750
Epoch 5/20
100/100 [==============================] - 35s 349ms/step - loss: 0.2135 - accuracy: 0.9112 - val_loss: 0.2840 - val_accuracy: 0.8750
Epoch 6/20
100/100 [==============================] - 35s 351ms/step - loss: 0.1806 - accuracy: 0.9269 - val_loss: 0.3770 - val_accuracy: 0.8750
Epoch 7/20
100/100 [==============================] - 35s 347ms/step - loss: 0.1905 - accuracy: 0.9212 - val_loss: 0.2901 - val_accuracy: 0.8750
Epoch 8/20
100/100 [==============================] - 35s 349ms/step - loss: 0.2033 - accuracy: 0.9219 - val_loss: 0.2564 - val_accuracy: 0.8750
Epoch 9/20
100/100 [==============================] - 35s 348ms/step - loss: 0.1852 - accuracy: 0.9312 - val_loss: 0.5002 - val_accuracy: 0.7500
Epoch 10/20
100/100 [==============================] - 35s 348ms/step - loss: 0.1951 - accuracy: 0.9200 - val_loss: 0.2487 - val_accuracy: 0.8125
Epoch 11/20
100/100 [==============================] - 36s 357ms/step - loss: 0.2043 - accuracy: 0.9244 - val_loss: 0.3672 - val_accuracy: 0.8750
Epoch 12/20
100/100 [==============================] - 35s 349ms/step - loss: 0.1841 - accuracy: 0.9256 - val_loss: 0.2519 - val_accuracy: 0.9375
Epoch 13/20
100/100 [==============================] - 35s 354ms/step - loss: 0.1897 - accuracy: 0.9256 - val_loss: 0.3484 - val_accuracy: 0.9375
Epoch 14/20
100/100 [==============================] - 35s 351ms/step - loss: 0.1760 - accuracy: 0.9294 - val_loss: 0.3650 - val_accuracy: 0.8125
Epoch 15/20
100/100 [==============================] - 35s 349ms/step - loss: 0.1808 - accuracy: 0.9281 - val_loss: 0.3964 - val_accuracy: 0.8750
Epoch 16/20
100/100 [==============================] - 35s 345ms/step - loss: 0.1567 - accuracy: 0.9331 - val_loss: 0.4059 - val_accuracy: 0.8750
Epoch 17/20
100/100 [==============================] - 35s 346ms/step - loss: 0.1762 - accuracy: 0.9281 - val_loss: 0.3431 - val_accuracy: 0.8750
```

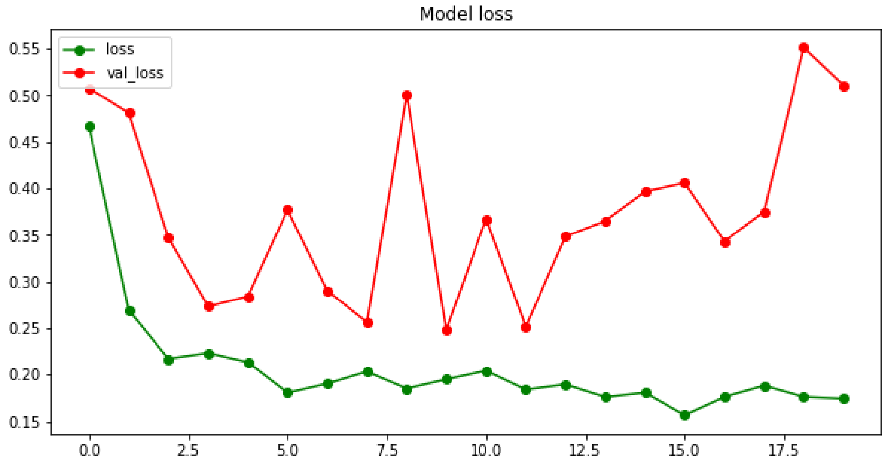
```
Epoch 18/20
100/100 [==============================] - 36s 356ms/step - loss: 0.1883 - accuracy: 0.9194 - val_loss: 0.3750 - val_accuracy: 0.8125
Epoch 19/20
100/100 [==============================] - 35s 354ms/step - loss: 0.1762 - accuracy: 0.9256 - val_loss: 0.5511 - val_accuracy: 0.7500
Epoch 20/20
100/100 [==============================] - 35s 349ms/step - loss: 0.1743 - accuracy: 0.9337 - val_loss: 0.5106 - val_accuracy: 0.8750
CPU times: user 12min 5s, sys: 7.03 s, total: 12min 12s
Wall time: 12min 21s
```

Visualize performance of model3

```
plt.figure(figsize = (10, 5))
plt.title("Model accuracy")
plt.plot(history.history["accuracy"], "go-")
plt.plot(history.history["val_accuracy"], "ro-")
plt.legend(["accuracy", "val_accuracy"])
plt.show()
```




```
plt.figure(figsize = (10, 5))
plt.title("Model loss")
plt.plot(history.history["loss"], "go-")
plt.plot(history.history["val_loss"], "ro-")
plt.legend(["loss", "val_loss"])
plt.show()
```



Evaluate model3 on the test set

```
#scores = model1.evaluate_generator(test_set)
#print("\n%s: %.3f%%" % (model1.metrics_names[0], scores[0]*100))
#print("\n%s: %.3f%%" % (model1.metrics_names[1], scores[1]*100))

test_loss, test_accuracy = model3.evaluate(test_set, steps = 50)
print("The testing accuracy is: ", test_accuracy * 100, "%")
print("The testing loss is: ", test_loss * 100, "%")
```

```
50/50 [==============================] - 6s 123ms/step - loss: 0.2887 - accuracy: 0.8894
The testing accuracy is:  88.94230723381042 %
The testing loss is:  28.872865438461304 %
```

Prediction of a new image

```
%%time

new_image_path = "/kaggle/input/pneumonia-lungs/download.jpg"
test_image = image.load_img(new_image_path, target_size = (224, 224))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
#test_image = np.reshape(test_image, (1, 224, 224, 3))
test_image = test_image / 255

#___________________________________________________________________

result = model3.predict(test_image)
train_set.class_indices
if result[0][0] == 0:
    prediction = "N O R M A L"
```

```
else:
    prediction = "P N E U M O N I A"

print(prediction)
```

```
P N E U M O N I A
CPU times: user 900 ms, sys: 7.97 ms, total: 908 ms
Wall time: 942 ms
```

Save the model to disk

```
model3.save("my_pneumonia_detection_model.h5")
print(colored("Model3 was succesfully saved to disk...", color = "green", attrs = ["bold", "dark"]))
```

```
Model3 was succesfully saved to disk...
```

Some time later you may need that model to use

```
model_loaded = tensorflow.keras.models.load_model("/kaggle/working/my_pneumonia_detection_model.h5")
model_loaded.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50v2 (Functional)      (None, 2048)              23564800
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 33
=================================================================
Total params: 23,837,441
Trainable params: 272,641
Non-trainable params: 23,564,800
_________________________________________________________________
```

Use loaded model to predict new image

```
def image_prediction(new_image_path):
    test_image = image.load_img(new_image_path, target_size = (224, 224))
    test_image = image.img_to_array(test_image)
    #test_image = np.reshape(test_image, (224, 224, 3))
```

```
    test_image = np.expand_dims(test_image, axis = 0)
    test_image = test_image / 255.0
    model_loaded = tensorflow.keras.models.load_model("working/my_pneumonia_detection_model.h5"
    prediction = model_loaded.predict(test_image)
    test_image_for_plotting = image.load_img(new_image_path, target_size = (224, 224))
    plt.imshow(test_image_for_plotting)
    if(prediction[0] > 0.5):
        statistic = prediction[0] * 100
        print("This image is %.3f percent %s"% (statistic, "P N E U M O N I A"))
    else:
        statistic = (1.0 - prediction[0]) * 100
        print("This image is %.3f percent %s" % (statistic, "N O R M A L"))


# call and use the function
image_prediction("pneumonia-lung-image-for-test/image.webp")
```

```
This image is 100.000 percent P N E U M O N I A
```